Instituto de Ciências Matemáticas e de Computação

ISSN -

Otimização por Decomposição

MARCIO KASSOUF CROCOMO
ALEXANDRE CLÁUDIO BOTAZZO DELBEM

Nº

RELATÓRIO TÉCNICO

São Carlos
OUTUBRO/2011

Resumo

Neste trabalho é proposta uma nova classe de algoritmos de otimização, chamada de Otimização por Decomposição (ODec). Estes algoritmos dividem-se em 3 etapas, sendo a primeira a criação de um conjunto inicial de soluções, a segunda responsável por identificar os blocos de variáveis relacionadas do problema sendo otimizado, e a terceira reponsável por buscar soluções ótimas do problema. Um algoritmo pertencente a esta classe é proposto, adicionando a este relatorio uma explicação das técnicas envolvidas na construção do mesmo. Experimentos comparando a técnica implementada com um EDA são apresentados, mostrando: (i) a eficiencia da técnica proposta e (ii) uma limitação da técnica, com relação a escalabilidade da mesma.

Agradecimento: FAPESP (processo nº 2010/01634-5)

Abstract

This document presents a new class of optimization algorithms, called "Optimization by Decomposition"(ODec). These algorithms are composed of 3 steps: the first is the creation of a random set of solutions; the second is the identification of the related variables that compose the problem being optimized; and the final step is the search for the optimal solution(s) of the problem. The document contains explanation of techniques for a proposed ODec algorithm. Also, experiments were conducted comparing the method with an Estimation of Distribution Algorithm, showing: (i) the efficiency of the proposed technique and (ii) a limitation of this technique, related to its scalability.

Sumário

Capítulo 1

Introdução

Vários problemas do mundo real são representados como problemas de otimização, nos quais um conjunto de parâmetros deve ser ajustado para que uma pontuação máxima seja atingida. Entre esses problemas, existem os complexos e de larga escala que, em geral, não podem ser resolvidos eficientemente por técnicas determinísticas de otimização. Como exemplo, podem ser citados problemas de redes elétricas (Mansour et al., 2010; Hadi e Rashidi, 2005) e predição de estruturas de proteínas (Lima, 2006; Cotta, 2003).

Classes de Algoritmos, como os Algoritmos Evolutivos (EAs, do inglês *Evolutionary Algorithms*), surgiram como solução para estes casos, buscando encontrar soluções que resultem na melhor pontuação possível. Destes algoritmos, os Algoritmos Genéticos (GAs, do inglês *Genetic Algorithms*) destacam-se ao resolver de forma eficiente diversos problemas (Ogata, 2006; Moura et al., 2010; Pessin et al., 2010). No entanto, os AGs começaram a se mostrar insuficientes para resolver com o desempenho desejado algumas categorias de problemas (Rana e Whitley, 1998; Harik et al., 1999).

Desta forma, surge um novo ramo da Computação Evolutiva, os Algoritmos de Estimação de Distribuição (EDAs, do inglês *Estimation of Distribution Algorithms*), que buscam criar um modelo probabilístico que represente as melhores soluções encontradas para o problema. Este modelo é então usado para direcionar a busca por novas e melhores soluções. Uma das abordagens utilizadas para se criar tal modelo probabilístico, é a identificação de blocos de construção (BBs, do inglês *Building Blocks*). EDAs como o Algoritmo Genético Compacto Extendido (ECGA, do inglês *Extended Compact Genetic Algorithm*) (Harik et al., 2006) e o algoritmo de Otimização Bayesiana com Detecção de Comunidades (OBDC) (Crocomo e Delbem, 2011) utilizam essa estratégia, ao identificar BBs a cada etapa do algoritmo iterativo.

Este documento propõe uma nova categoria de algoritmos: os algoritmos de Otimização por Decomposição (ODec). Enquanto os EDAs citados buscam identificar os BBs do problema a cada etapa do algoritmo iterativo, os algoritmos ODec buscam encontrar os BBs uma única vez, e então usam essa informação junto a um algoritmo de busca para encontrar a melhor solução possível.

Este relatório introduz o conceito de BBs no Capítulo 2 e, então, explica o funcionamento dos algoritmos ODec no Capítulo 3. Ainda neste capítulo, é proposto um algoritmo ODec, chamado Otimização Direta (ODir). Os capítulos seguintes explicam as técnicas utilizadas na contrução do ODir: K2 (Capítulo 4) e *Fast Algorithm* (Capítulo 5). Por fim, são apresentados os experimentos realizados com este algoritmo no Capítulo 6 e as Conclusões no Capítulo 7.

Capítulo 2

Blocos de Construção

Ao se analisar a dificuldade de problemas, é preciso verificar como essa varia em escala, isto é, com o crescimento do tamanho do problema. É importante caracterizar também o tamanho do espaço de busca. Isso depende não somente do número de variáveis do problema, como em geral é apresentado em análises de complexidade computacional, mas também da cardinalidade das variáveis. Por exemplo, ao trabalhar com variáveis binárias, tem-se um espaço de busca de tamanho 2^n, onde n é o número de variáveis. Assim, para variáveis χ-árias, o espaço de busca possui tamanho χ^n.

Embora o tamanho do espaço de busca seja um fator muito importante para se determinar a dificuldade de um problema, limitantes superiores de complexidade significativamente menores que χ^n podem ser encontrados. Em (Goldberg, 2002) é verificado que dois fatores muito importantes são: A quantidade m de Blocos de Construção (BBs, do inglês *Building Blocks*) e o número k de variáveis que compõem cada BB.

Para ilustrar a importância desses fatores, considere três problemas distintos, representados na Figura 2.1, todos referentes a um conjunto de 5 variáveis binárias (5 *bits*). Os problemas ilustrados são dados por funções unitárias, o que significa que a entrada para a função é um valor u que corresponde à quantidade de variáveis binárias que possuem o valor '1'. Os problemas são os seguintes:

1. Problema UmMax (Goldberg, 2002): consiste em maximizar a quantidade de variáveis iguais a 1. O *fitness* é a quantidade de variáveis com valor 1;

2. Problema de senha: *fitness* é 0 para todas as soluções, exceto para o caso em que todas as variáveis são 1, para o qual o *fitness* possui valor 5;

3. Problema armadilha (Goldberg, 2002): o *fitness* cai com o aumento de u, exceto para o maior u que possui o valor de *fitness* máximo como, por exemplo, a seguinte função armadilha de 5 *bits*:

$$ftrap5(u) = \begin{cases} 4 - u & se\ u < 5, \\ 5 & caso\ contrário. \end{cases} \tag{2.1}$$

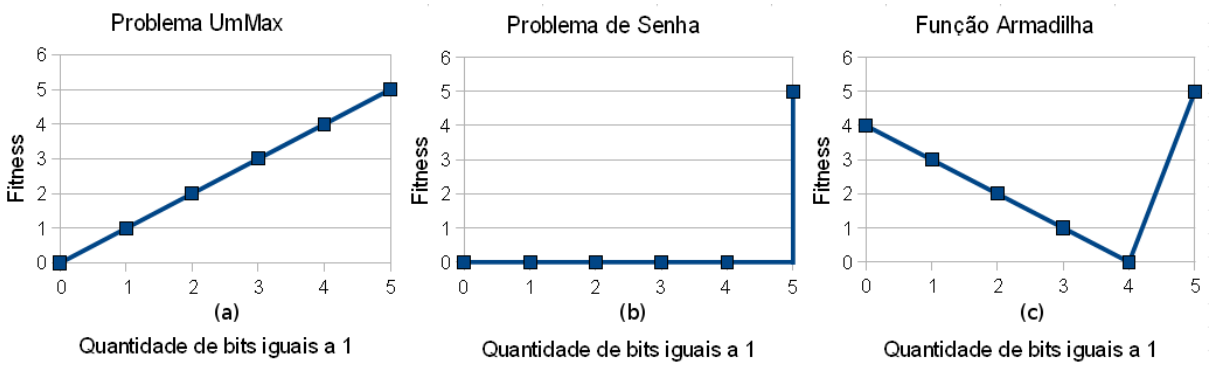

Figura 2.1: Problemas de funções unitárias: (a)UmMax, (b)Problema de senha, (c)Função armadilha.

Ao modificar o valor de um único *bit* de entrada no problema UmMax, pode-se saber se a alteração gerou uma solução mais próxima do ótimo global apenas verificando se a alteração aumentou ou diminuiu o *fitness*. Observe que isso independente do valor de qualquer uma das outras variáveis. Devido a esse fato, pode-se considerar que cada variável é, isoladamente, um BB. Assim, este problema é composto por 5 BBs de tamanho um, que são 5 subproblemas independentes. No problema de senha ou no problema de armadilha, para se saber se a modificação de um *bit* gerou uma solução mais próxima do ótimo global, é preciso saber os valores dos outros 4 *bits* de entrada. Em outras palavras, os 5 *bits* são dependentes entre si, formando um BB. Portanto, esses dois problemas possuem um único BB de tamanho 5. A partir deste ponto, é possível observar que, embora todos os problemas tenham o mesmo espaço de busca: 2^5. Os problemas com BBs de tamanho maior são mais difíceis.

No problema UmMax, algoritmos de busca local (Hoos e Stützle, 2004) são suficientes para se encontrar o ótimo global do problema (11111). No entanto, para o problema de senha, qualquer meta-heurística (Talbi, 2009) é no máximo tão eficiente quanto uma busca exaustiva ou aleatória, uma vez que não existe informação em nenhum local do espaço de busca que possa guiar o algoritmo para o ótimo global (com exceção do próprio ótimo global). Por outro lado, no problema armadilha, há informação. Com isso, meta-heurísticas que utilizem alguma informação local para orientar a busca, tendem a orientar a busca para soluções com poucos 1s, podendo a solução 00000 (ótimo local da $ftrap5$ dominar a população e impedir a descoberta do ótimo global (11111). Portanto, para essas metaheurísticas, a informação existente é na verdade desinformação, uma armadilha.

Uma forma de construir problemas mais difíceis que o problema armadilha é simplesmente pelo somatório de subproblemas em que cada um é uma função armadilha. Como exemplo, considere um problema composto pelo somatório de 3 funções $ftrap5$. Nesse exemplo, um cromossomo é formado então por um conjunto de 15 *bits*, os quais são particionados em 3 subconjuntos de 5 *bits* e uma $ftrap5$ é aplicada a cada subconjunto. O *fitness* do cromossomo é a soma dos resultados obtidos para cada $ftrap5$. Dessa forma, o problema é formado por 3 BBs de tamanho 5 ($k = 5$ e $m = 3$). Como discutido anteriormente, em funções armadilha, a busca dentro de um BB, é induzida para ótimos locais diferente do ótimo global. Uma alternativa apresentada em (Goldberg, 2002) para vencer as armadilhas é que o processo de busca não

ocorra dentro dos BBs, mas entre os BBs. Em outras palavras, parte-se do princípio de que a população inicial do algoritmo possua pelo menos uma representação de cada instância possível de um BB, então o operador de recombinação utilizado não mais tem como objetivo combinar as variáveis do problema, mas sim combinar os BBs existentes na população. Reforça-se que, para isso, é necessário que a população inicial tenha uma grande diversidade de soluções provendo pelo menos uma instância ótima de cada BB na população de forma que essas instâncias possam ser combinadas gerando a solução ótima do problema.

Foi demonstrado em (Goldberg, 2002) que a Equação 2.2 estima adequadamente o tamanho da população inicial para que essa amostre com alta probabilidade, todas as possíveis instâncias de um BB.

$$n = \chi^k(k * \ln\ \chi + \ln\ m), \tag{2.2}$$

em que n é o tamanho da população, χ a cardinalidade das variáveis, k o tamanho do BB e m a quantidade de BBs. Como k é expoente positivo na Equação 2.2 e os demais termos não são exponenciais, χ^k domina a estimativa de n conforme k cresce.

A Figura 2.2 ilustra como o tamanho da população inicial necessária para que todas as possíveis instâncias dos BBs sejam representadas na população variam com k e m. As linhas contínuas são obtidas pela Equação 2.2; enquanto os pontos marcados foram obtidos experimentalmente através dos passos explicados abaixo, reproduzidos a partir dos experimentos relatados em 2.2

1. Crie uma população inicial com 0 indivíduo;
2. Enquanto houverem instancias não representadas de algum BB na população, inclua um novo individuo aleatório;
3. Retorne o tamanho da população final (que contém todas as instâncias de cada BB).

Cada ponto representado no gráfico é obtido para um dado valor de k e de m, e é encontrado a partir de 30 execuções dos passos acima, sendo o ponto exibido dado pela média dos resultados encontrados. Dessa forma, ilustra-se teórica e experimentalmente como a complexidade de um problema aumenta conforme k e m aumentam. Além disso, a Figura 2.2 mostra claramente que o tamanho do BB é o fator mais influente a ser considerado.

No entanto, uma amostra inicial grande (contendo amostras com o ótimo global de cada subproblema) não é o suficiente para que o ótimo global do problema seja construído. É necessário identificar previamente quais são as variáveis relacionadas que compõem os BBs, para que se possa combinar instâncias de BBs presentes em diferentes indivíduos ao longo das gerações até atingir a combinação correspondente ao ótimo global. Dentre os EDAs que trabalham com esse princípio, podemos citar:

***Algoritmo Genético Compacto Extendido*:**

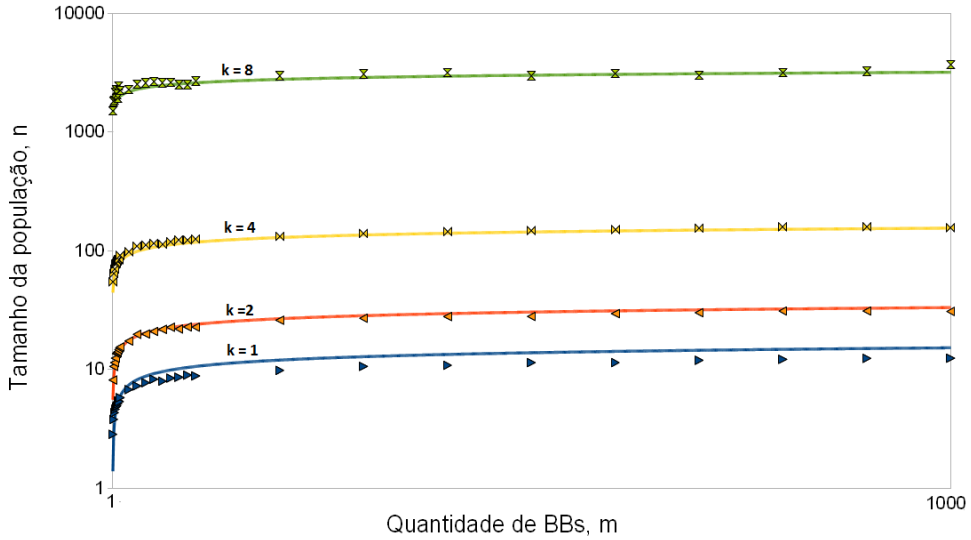

Figura 2.2: Tamanho da população inicial necessária para que exista pelo menos uma de cada possível instância de um BB.

O ECGA (Harik et al., 2006) busca encontrar as variáveis relacionadas através de um processo chamado de *linkage learning*. Esse método considera que as variáveis relacionam-se em grupos (BBs) sem que estes sejam sobrepostos, isto é, considera que não existe uma ou mais variáveis pertencentes a dois grupos distintos. Uma vez obtido um modelo que representa os BBs para um problema, é possível usar operadores reprodutivos competentes (Goldberg, 2002) que não misturam valores de variáveis de instâncias de um BB de indivíduos diferentes ao combinar duas ou mais soluções. Observe que, sem o modelo de BBs, o operador de recombinação pode desagrupar combinações promissoras de valores de variáveis se o ponto de corte for interno a um BB, ou seja, pode destruir uma solução ótima de um subproblema.

***Algoritmo de Otimização Bayesiana*:**

A ideia central do BOA (Pelikan, 2005; Pelikan et al., 1999) é a utilização de Redes Bayesianas (BNs, do inglês *Bayesian Networks*) (Niedermayer, 2009) para representação de informações a respeito das dependências entre variáveis. As variáveis são representadas como vértices em uma rede (um grafo) e a existência de uma aresta entre dois vértices significa que o valor de uma dessas variáveis depende do valor da outra. Em outras palavras, as componentes conexas (Diestel, 2005) da rede podem ser vistas como BBs. Diferentemente do ECGA, a BN é capaz de representar sobreposição de BBs. A partir de uma BN e de indivíduos selecionados da população, é possível gerar novos indivíduos que irão compor a nova geração.

***Algoritmo de Otimização Bayesiana com Detecção de Comunidades*:**

O algoritmo de Otimização Bayesiana com Detecção de Comunidades (OBDC) (Crocomo e Delbem, 2011) é uma extensão do BOA, que utiliza técnicas de detecção de estruturas de comunidades para identificar grupos de variáveis mais fortemente relacionadas tendo como entrada a BN gerada. As comunidades encontradas desta forma, representam os BBs do problema, e é utilizada para melhorar o modelo obtido, ao reforçar as ligações entre as variáveis pertencentes a uma mesma comunidade na BN. Este algoritmo encontra-se detalhado em (Crocomo e Delbem, 2011).

Capítulo 3

Otimização por Decomposição

Os resultados promissores obtidos com o algoritmo OBDC (Crocomo e Delbem, 2011) motivaram a ideia de uma categoria de algoritmos chamados de ODec. EDAs como o ECGA e o OBDC buscam encontrar a cada iteração do algoritmo, um modelo que descreve o problema ao identificar os BBs. É importante lembrar que os EDAs substituem a aplicação de operadores reprodutivos (Bäck et al., 2000; Jong, 2006; Fogel, 2005) como os de crossover e de mutação pelos seguintes passos:

1. Construção de um modelo probabilístico multivariado que representa de forma sintética os indivíduos selecionados;

2. Geração de novos indivíduos, baseada no modelo construído.

O Passo 1 acima, consiste em criar um modelo que identifique conjuntos de variáveis relacionadas. Uma vez criado tal modelo, é possível usar operadores reprodutivos competentes, que não destroem um BB apropriado durante sua aplicação. Observe que, sem o modelo de BBs, o operador de recombinação pode desagrupar variáveis se o ponto de corte for interno a um BB (Goldberg, 2002). Note que os Passos 1 e 2 são realizados diversas vezes por um EDA convencional, construindo um modelo a cada geração do EDA.

A ideia de um algoritmo ODec, é a de usar um algoritmo para detecção dos BBs do problema uma única vez e, então, aplicar uma busca separada para cada um destes BBs. Esses algoritmos são baseados em 3 passos, ilustrados na Figura 3.1, e detalhados abaixo:

1. Gerar uma população inicial grande o suficiente para conter pelo menos uma instância de cada BB;

2. Utilizar um algoritmo para identificar os BBs com grande precisão;

3. Realizar uma busca, utilizando a informação obtida na identificação dos BBs.

Figura 3.1: Ilustração do funcionamento de um algoritmo ODec.

Para que exista uma grande probabilidade de que os BBs sejam corretamente encontrados, deve-se possuir uma amostra inicial de soluções que seja grande o suficiente para que cada instância possível de um BB esteja representada ao menos uma vez. A Equação 2.2 (ver Seção 2) é um limitante inferior do tamanho da população gerada no Passo 1 que garante a amostragem adequada de cada instância de BB.

Durante o Passo 2, é utilizado um algoritmo para a detecção dos BBs. Normalmente, uma avaliação da população obtida em 1 é realizada, e então essa população, junto com sua avaliação, servem de entrada para um algoritmo responsável por identificar as variáveis relacionadas. Possíveis estratégias a serem utilizadas nesse passo são as etapas de identificação dos BBs utilizadas por EDAs, como a composição do K2 com o FA, utilizada no OBDC, ou a utilização da métrica de complexidade combinada com uma busca gulosa, como realizado no ECGA.

Admitindo o uso de um eficiente algoritmo no Passo 2, sobre uma amostra representativa das soluções (calculada no Passo 1), tem-se a partir deste ponto a estrutura dos BBs que compõe o problema inicial. O Passo 3 realiza uma busca que utiliza a informação dos BBs que compoem o problema, obtida no passo 2. Exemplo de algoritmos a serem utilizados nessa etapa são: (i) busca gulosa, (ii) EAs e (iii) EDAs. É importante observar que o modelo proposto não é necessariamente um procedimento iterativo, dependendo da busca utilizada no Passo 3.

3.1 Otimização Direta

Como exemplo de um algoritmo ODec, é proposto neste trabalho o algoritmo chamado Otimização Direta (ODir). Esse algoritmo usa para o Passo 2 o algoritmo K2 (Seção 4) para encontrar uma BN a partir da população gerada no Passo 1. Em seguida, um algoritmo de detecção de estrutura de comunidades (o FA, ver Seção 5) é usado para identificar os BBs, tendo a BN como entrada. A combinação desses dois métodos é o mesmo procedimento utilizado no algoritmo OBDC para a identificação dos BBs em cada iteração, com a diferença de que agora não são mais reforçadas as ligações entre as variáveis dentro de uma mesma comunidade. Além disso, não é mais de interessante calcular uma tabela de probabilidades como feito pelo BOA ou pelo OBDC, a fim de se gerar os novos indivíduos. Em outras palavras, o interesse de utilizar estes dois métodos é apenas o de compreender melhor o problema, identificando seus BBs. Por este motivo, após a obtenção dos BBs que compõem o problema, a BN é descartada.

O Passo 3 realiza uma busca gulosa para encontrar a melhor instância em cada um dos BBs. A busca gulosa é adequada ao admitir que cada BB possui tamanho relativamente pequeno. Observe que, conforme mostra o Capítulo 2, não é possível garantir com EAs que soluções

ótimas de cada BB sejam encontradas para problemas contendo BBs de tamanho não pequenos. No entanto, para os casos com BBs pequenos, existe uma alta probabilidade de que a melhor solução seja encontrada pelo ODir.

A busca gulosa é realizada partindo da melhor solução encontrada na população. A Figura 3.2 ilustra o caso em que um cromossomo de 4 *bits* é selecionado. No exemplo, existem dois BBs compostos por 2 *bits* cada. Na primeira etapa, são verificadas as pontuações de cada solução considerando todas as possíveis instâncias do primeiro BB. A melhor solução é selecionada e o procedimento repete-se, testando as pontuações de cada solução formada considerando todas as possíveis instâncias do segundo BB. Quando as melhores instâncias para cada BB forem encontradas dessa maneira, é gerada a solução final pelo algoritmo.

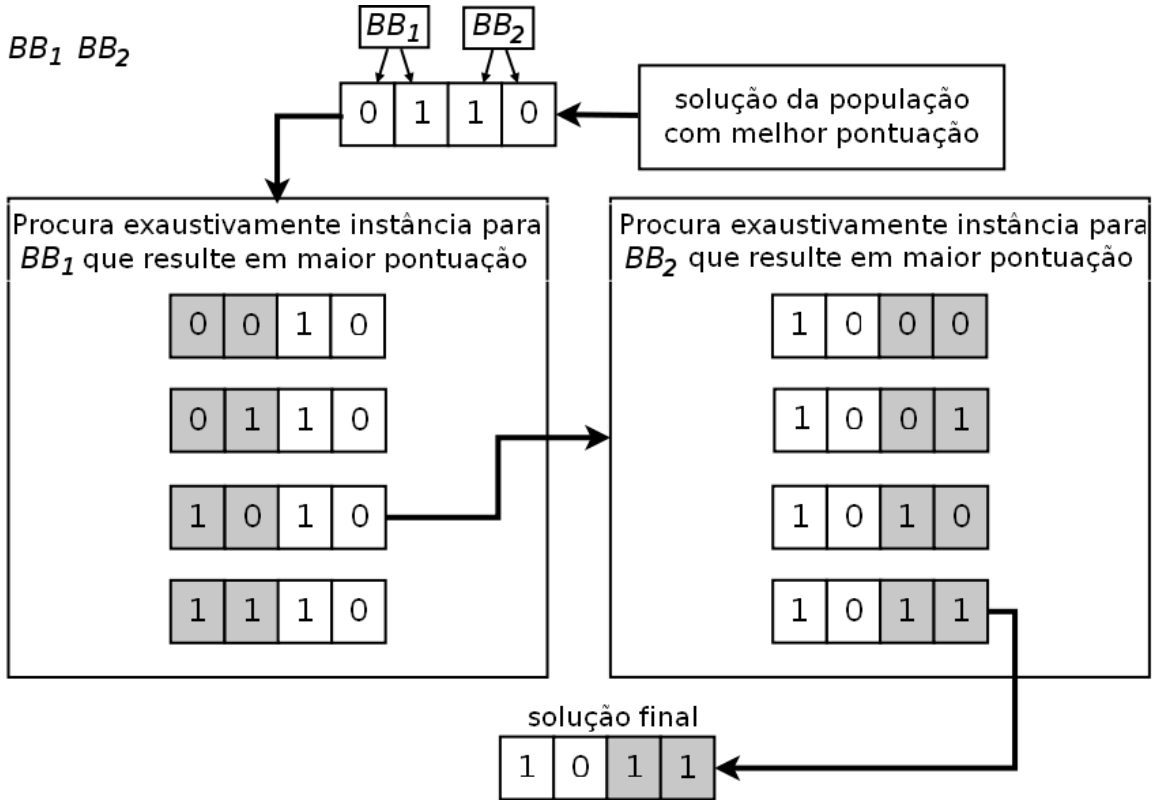

Figura 3.2: Busca gulosa no algoritmo ODir.

O Algoritmo 1 apresenta o pseudocódigo do ODir.

Algoritmo 1: Otimização Direta.

Entrada: Uma função de avaliação para soluções do problema, e o tamanho da população (tamanhoPopulação) a ser utilizada.

Saída: A melhor solução encontrada para o problema.

1 *população* = IniciaPopulação($tamanhoPopulacao$);
2 AvaliaPopulação();
`// Aplique o algoritmo K2 para encontrar a Rede Bayesiana` BN
3 BN = K2(*população*);
`// Aplique o Fast Algorithm pra identificar os BBs`
4 FA(BN);
5 *solução* = melhorSolução(*população*);
6 **para** *cada* BB_j *encontrado* **faça**
`// Aplique uma busca exaustiva para cada` BB_j
7 *instânciaBBj* = buscaExaustiva(BB_j);
`// Adicione a instância encontrada em uma solução final`
8 adicioneNaSoluçãoFinal(*soluçãoFinal*,*instânciaBBj*);
9 **fim**
10 retorne *soluçãoFinal*;

Os Capítulos 4 e 5 explicam as técnicas utilizadas na execução do algoritmo ODir: O algoritmo K2, responsável pela obtenção da BN, e o *Fast Algorithm* (FA), que é uma técnica de detecção de estrutura de comunidades.

Capítulo 4

Algoritmo K2

Há dois componentes básicos no algoritmo que constrói a topologia de uma BN (Pearl, 1988):

1. A métrica de pontuação;
2. O procedimento de busca.

A métrica de pontuação avalia a qualidade com que a rede modela os dados. O conhecimento prévio sobre o problema também pode ser incorporado na métrica. É importante destacar que encontrar a melhor BN (que possui a melhor pontuação dada pela métrica) para um conjunto de variáveis é um problema NP (Santos, 2007). Dessa forma, busca-se na prática um procedimento que construa uma rede com o valor de métrica de pontuação tão alto quanto possível em um tempo computacional aceitável (observe que uma nova BN deve ser construída a cada geração para cada novo conjunto selecionado). A pontuação de uma BN (B_s) dado um conjunto de dados (D), chamada de $P(B_s, D)$, pode ser dada pela Equação 4.1.

$$P(B_s, D) = P(B_s) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!, \tag{4.1}$$

em que:

- $P(B_s)$ é um fator utilizado quando se tem uma informação prévia sobre a qualidade da rede. Quando não há tal informação, este valor é 1 (valor usado neste trabalho);
- n é número de variáveis X_i;
- q_i é o número de possíveis combinações encontradas na base de dados para os valores dos pais de X_i;
- R_i é o número de possíveis valores associados à variável X_i;
- N_{ijk} é o número de casos em D, com $X_i = v_{ik}$ e $\pi_i = w_{ij}$;

- v_{ik} é o k-ésimo possível valor para a variavel X_i;
- π_i é o conjunto de pais da variável X_i;
- w_{ij} é a j-ésima possível instância para os valores de π_i;
- N_{ij} é o número de ocorrências que possuem o conjunto de pais de X_i instanciados como w_{ij}. Dado por: $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

O algoritmo K2 utiliza um método de busca gulosa (*greedy search*, (Cook et al., 1997)) para maximizar P(Bs|D). Assim, a contribuição de um vértice i da rede no valor da pontuação $P(Bs, D)$ (efeito local) depende do conjunto de pais deste vértice (i, π_i) e pode ser explicitado pela Equação 4.1, definida a partir da Equação 4.2. Para melhor compreender esta equação e as variáveis envolvidas, um exemplo pode ser encontrado no Apêndice A.

$$g(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!. \tag{4.2}$$

O pseudocódigo do algoritmo K2 (descrito no Algoritmo 2, adaptado de (Cooper e Herskovits, 1992)) pode ser explicado como segue. Partindo de um grafo sem arestas, uma busca gulosa é utilizada para, a cada etapa, para escolher uma nova aresta que será inserida na rede. A modificação na pontuação da BN pela inserção desta aresta, é dada pela Equação 4.2. Esta equação torna-se uma complicação, pois, o uso de fatoriais produz números muito grandes, podendo causar *overflow* quando o algoritmo é executado. Portanto, deve-se utilizar a forma logarítmica da expressão, substituindo $g(i, \pi_i)$ por $log(g(i, \pi_i))$, conforme mostrado pela Equação 4.3.

$$log(g(i, \pi_i)) = \sum_{j=1}^{q_i} ln((r_i - 1)!) - ln((N_{ij} + r_i - 1)!) + \sum_{k=1}^{r_i} N_{ijk}!. \tag{4.3}$$

Algoritmo 2: Pseudocódigo do algoritmo K2.

Entrada: Um conjunto de n vértices ordenados, limite superior v (número máximo de pais por vértice), base de dados D, contendo t casos.

Saída: Para cada vértice, a identificação de seus pais.

```
1  para i = 1 até t faça
2      π_i = {};
       // Função calculada usando a fórmula logaritmica, dada
          pela Equação 4.3
3      P_old = g(i,π_i);
4      OKparaProceder = sim;
5      enquanto OKparaProceder e |π|<v faça
           // z aponta para o vértice que, entre todos os
              vértices que antecedem X_i dada a ordenação de
              entrada, maximiza g(i,π_i ∪ {z})
6          z = maxPred(X_i);
7          P_new = g(i,π_i ∪ {z});
8          se P_new > P_old então
9              P_old = P_new;
10             π_i = π_i ∪ z;
11         fim
12         senão OKparaProceder = não;
13     fim
14     Escreva("Vértice:"X_i, "Pais deste vértice:", π_i);
15 fim
```

Capítulo 5

Detecção de Estruturas de Comunidades

Devido ao fato de pessoas, em geral, dividirem-se em grupos de acordo com seus interesses, trabalho, idade e outras características, redes sociais normalmente encontram-se subdivididas em uma estrutura de comunidades (Newman e Girvan, 2004). Essas podem ser formalmente representadas por grupos de vértices de uma rede que são densamente conectados entre si, e esparsamente conectados com o resto da rede, que contém as outras comunidades. A Figura 5.1 ilustra uma possível estrutura de comunidades para uma rede.

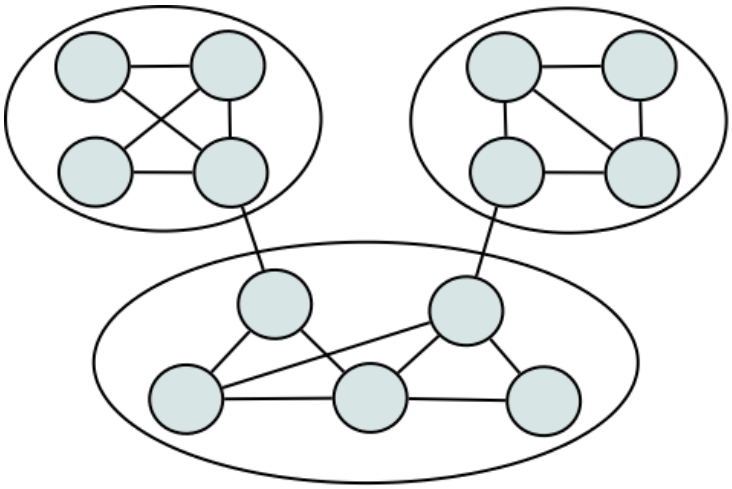

Figura 5.1: Exemplo de estrutura com três comunidades em uma rede.

Comunidades são de grande interesse, pois, geralmente correspondem a unidades comportamentais e funcionais. Devido a isso, ferramentas matemáticas e algoritmos têm sido pesquisados para detectar estruturas de comunidades na área de Redes Complexas (Donetti e Muñoz, 2004; Duch e Arenas, 2005). É importante ressaltar que algoritmos que verificam todas as possíveis estruturas de comunidade em uma dada rede são NP (Duch e Arenas, 2005) e, portanto, inviáveis de serem utilizados para redes grandes. Por isso, a literatura recente apresenta diversas novas técnicas que buscam encontrar a melhor estrutura de comunidade possível em um tempo aceitável.

Dessas técnicas, pode-se destacar as baseadas na métrica Q, definida por Newman (Newman e Girvan, 2004) que representa a modularidade de uma divisão em comunidades. Quanto maior o valor de Q, melhor a estrutura de comunidades encontrada. A partir da modularidade, diversos pesquisadores têm proposto o uso de algoritmos de otimização para encontrar estruturas de comunidades. Entre esses, podem-se destacar o *Fast Algorithm* (FA) (Newman e Girvan, 2004), o Adaptive Clustering (adClust) (Ye et al., 2008) e o Otimização Extrema (EO, do inglês *Extreme Optimization*) (Duch e Arenas, 2005).

5.1 *Fast Algorithm*

O FA é baseado na ideia de modularidade, que é uma métrica para qualificar as estruturas das comunidades (Newman e Girvan, 2004). Para isso, define-se o termo e_{ij} como a metade do número de arestas que conectam as comunidades i e j em relação ao total de arestas da rede. Dessa forma, $e_{ij} + e_{ji}$ corresponde ao total de arestas que conectam os vértices de ambas as comunidades. Já e_{ii} corresponde ao número de arestas dentro da comunidade i em relação ao total de arestas. Um algoritmo de detecção de comunidades deve maximizar a fração de arestas que conectam vértices de uma mesma comunidade, ou seja, maximizar $\sum_i e_{ii}$. Entretanto, essa medida não avalia bem a qualidade das comunidades, uma vez que seu valor máximo é facilmente atingido quando todos os vértices da rede pertencem à mesma comunidade. Para isso, mais um componente é utilizado: $a_i = \sum_j e_{ij}$, a fração das arestas conectadas a pelo menos um vértice da comunidade i. Com isso, é define-se o índice de modularidade Q ponderando entre ambas frações conforme mostra a Equação 5.1.

$$Q = \sum_i (e_{ii} - a_i^2). \tag{5.1}$$

De posse de um índice para quantificar a qualidade das comunidades, pode-se construir um algoritmo que otimize o valor de Q sobre todas as possíveis divisões da rede em comunidades. Foi desenvolvido em (Newman, 2004) um algoritmo guloso para esse fim. Inicialmente, esse algoritmo considera cada vértice da rede como uma comunidade. Repetidamente, as comunidades são agrupadas em pares considerando todos os pares possíveis. Então, o agrupamento com maior valor de Q é preservado. O Algoritmo 3 apresenta o funcionamento do FA.

Algoritmo 3: *Fast Algorithm*

Entrada: Uma rede.

Saída: Uma estrutura de comunidades para a rede.

1 Separe os vértices em n comunidades (uma por vértice);
2 **enquanto** *número de comunidades > 1* **faça**
3 Junte as duas comunidades i e j cuja aglomeração forneça o maior acréscimo (ou menor decréscimo) no valor da modularidade;
4 Salve a divisão resultante e sua modularidade;
5 **fim**
6 Retorne a divisão com a melhor modularidade encontrada;

É importante ressaltar que apenas comunidades que possuam pelo menos uma aresta ligando seus vértices podem ser possivelmente unidas pelo algoritmo. Isso limita a um máximo de p pares de comunidades, onde p é o número de arestas do grafo. A variação em Q ao se unir duas comunidades é dada pela equação 5.2.

$$\Delta Q = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j) \tag{5.2}$$

O valor inicial de e_{ij} é igual à metade do grau de cada vértice, uma vez que inicialmente cada comunidade é formada por apenas um vértice. Após a junção de duas comunidades os valores de e_{ij} devem ser atualizados. Por fim, vale destacar que o FA proposto em (Newman, 2003) quando aplicado em redes de pequena escala, possui um desempenho similar ao algoritmo proposto em (Newman e Girvan, 2004). Porém, para redes de larga-escala, o FA possui desempenho superior.

Capítulo 6

Experimentos

Para testar a eficiência das versões dos algoritmos ODir e OBDC (relatório deste algoritmo disponível em (Crocomo e Delbem, 2011)) implementados, foram realizados os seguintes testes: o conjunto de variáveis que se procura otimizar é representado por um *array* binário composto por m BBs de tamanho 5 (k=5) a serem aplicados em funções $ftrap5$ (ver Seção 2). Foram realizados 30 testes para cada tamanho de problema (ℓ) utilizado: 30, 60 e 120. O tamanho da população (n) utilizada para cada um desses casos foi, respectivamente, de 1300, 3000 e 9000 indivíduos (determinados empiricamente). Os aspectos comparados entre estas duas técnicas são os seguintes:

- Taxa de acerto de BBs, dada pelo número de BBs para os quais foram encontrados o máximo global, dividido pelo total de BBs do problema;
- Número de avaliações até que o critério de convergência tenha sido atingido. Esse critério é o mais utilizado para se medir a eficiencia de um EDA. É admitido que a operação mais custosa é a avaliação de cada solução. Quanto menor for o número de avaliações realizadas para se obter uma boa solução, mais eficiente é o algoritmo.

Os resultados mostrados nas Figuras 6.1e 6.2 foram todos obtidos realizando uma média de 30 experimentos. É possível notar que o algoritmo ODir foi capaz de encontrar as soluções ótimas para os testes com 30 e 60 variáveis assim como o algoritmo OBDC. No entanto, o número de avaliações requerido pelo algoritmo ODir foi muito menor, indicando uma maior eficiência. Por outro lado, embora o algoritmo tenha utilizado um número de avaliações muito menor para o caso com 120 variáveis, observa-se uma queda na taxa de acerto do algoritmo. Tal fato ocorre devido a dificuldade em identificar os BBs, que cresce com o número de possíveis variáveis a serem associadas. Como o ODir cria um modelo que associa as variáveis uma única vez, erros obtidos na criação desse único modelo não são corrigidos, resultando em uma menor taxa de acerto. O mesmo não ocorre com o OBDC, já que o mesmo cria modelos iterativamente, em cada geração do algoritmo. Dessa forma, erros obtidos em gerações anteriores podem ser corrigidos durante a execução do algoritmo, resultando em uma melhor escalabilidade.

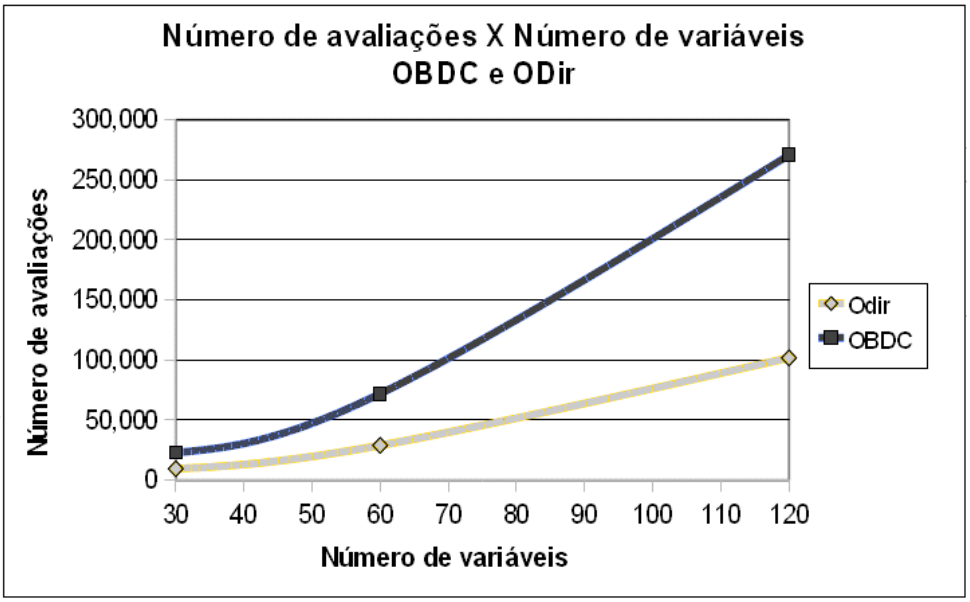

Figura 6.1: Número de avaliações necessárias para os algoritmos OBDC e ODir.

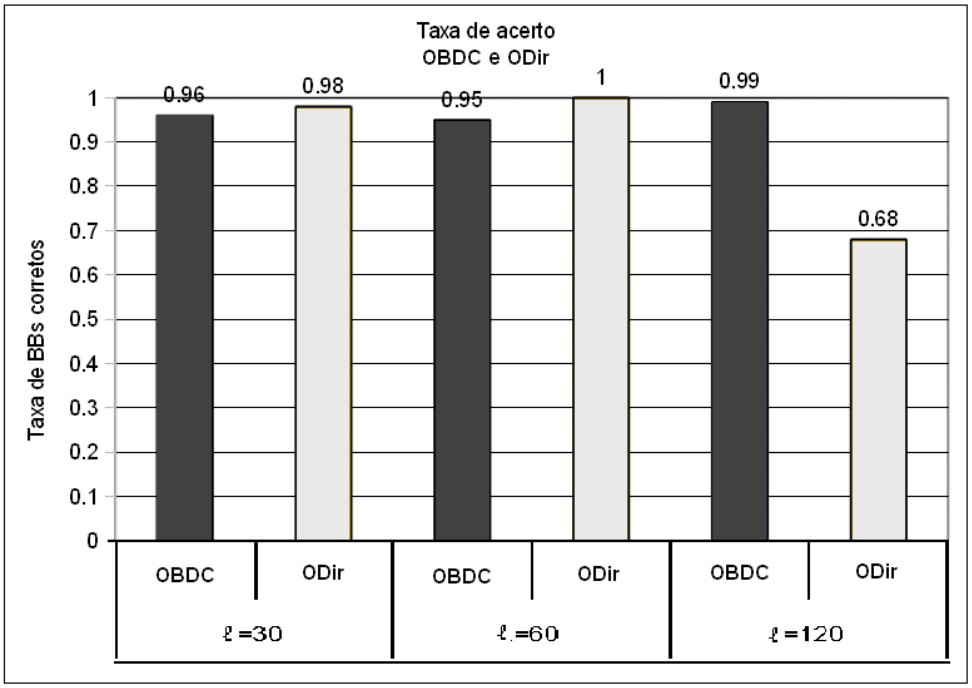

Figura 6.2: Proporção de BBs que atingiram o máximo global utilizando a técnica OBDC, com $v = 2$ e $v_f = 4$ e a técnica ODir, com os mesmos parâmetros. Os Resultados apresentados foram obtidos realizando a média de 30 testes, em problemas de tamanho 30, 60 e 120.

Capítulo 7

Conclusões

Neste trabalho, foi apresentada uma nova classe de algoritmos, chamada de ODec, e foi proposto um algoritmo pertencente a esta classe, o algoritmo ODir. Diferentemente dos EDAs tradicionais, os algoritmos ODec buscam identificar as variáveis relacionadas existentes no problema em um passo anterior ao algoritmo de busca pelo valor ótimo.

O algoritmo ODir proposto, após a identificação dos grupos de variáveis relacionadas no problema, utiliza uma simples busca gulosa para encontrar o valor ótimo. A utilização dessa busca gulosa limita os problemas resolvidos por este algoritmo à classe dos sem sobreposição de BBs. Apesar dessa restrição, o universo de problemas computacionalmente complexos que podem ser resolvidos é relativamente grande, podendo também ser adequada a aproximação para problemas mais complexos.

A utilização da busca gulosa neste algoritmo, garante que a solução ótima seja encontrada desde que os blocos de construção identificados sejam corretos. Portanto, a maior dificuldade do algoritmo encontra-se na etapa de identificação de BBs, que é um problema não trivial. Para tanto, o algoritmo ODie utiliza a construção de uma BN junto com um método de detecção de estrutura de comunidades.

Os resultados são motivadores, pois mostram que a abordagem utilizada pelos algoritmos ODec possibilita que problemas que não possuam sobreposição de BBs sejam resolvidos com uma eficiência significativamente maior que a de outros EDAs, como o OBDC. Experimentos futuros deverão ser realizados com uma nova implementação do algoritmo ODir, feita ao adaptar o código do BOA, criado pelos autores originais da técnica, comparando então os resultados dessas duas técnicas.

Embora promissores, os resultados neste trabalho já identificam uma limitação do algoritmo ODir, apontada pelos experimentos realizados: a escalabilidade. Para resolver essa limitação, uma possível continuação deste trabalho é o estudo de técnicas de reamostragem para que a partir da população inicial, outras populações sejam reamostradas, possibilitando a montagem de diferentes modelos que identifiquem os BBs. Exemplos de técnicas de reamostragem são o *bootstrap* e o *jacknife* (Efron e Gong, 1983). Como é esperado que os erros ao associar certo conjunto de variáveis estejam presentes em apenas alguns dos modelos de BBs criados,

é possível identificar os conjuntos mais comuns (mais verossímeis) de variáveis associadas, gerando um modelo mais confiável para o problema. Assim, seria melhorada a escalabilidade do algoritmo, sem prejudicar sua eficiência, uma vez que o número de avaliações necessárias continuaria o mesmo. A ideia de utilizar técnicas de reamostragem para auxiliar algoritmos de otimização já foi utilizada com exito na técnica *Smart Sampling* (Melo e Delbem, 2009), desenvolvida no Laboratório de Computação Reconfigurável (LCR).

Além do algoritmo ODir proposto neste relatório, outros algoritmos que se encaixam na classe ODec já foram desenvolvidos no Laboratório de Computação Reconfigurável (LCR), apresentando resultados relevantes. Estes algoritmos são: (1) o algoritmo Filogenético (PhyGA) (Vargas e Delbem, 2009), (2) o algoritmo Filogenético Multiobjetivo (mo-PhyGA) (Martins et al., 2011) e (3) o algoritmo Filogenético com Evolução Diferencial (PhyDE) (de Melo et al., 2011). Sendo que o PhyDE utiliza evolução diferencial na etapa de otimização, não limitando os problemas trataveis a problemas com BBs sem sobreposição, como ocorre quando a busca gulosa é utilizada. Possíveis continuações deste trabalho envolvem o desenvolvimento de outros algoritmos ODec.

Referências Bibliográficas

BÄCK, T.; FOGEL, D.; MICHALEWICZ, Z., eds. *Evolutionary computation 1: Basic algorithms and operators*. Bristol: Institute of Physics Publishing, 339 p., 2000.

COOK, W. J.; CUNNINGHAM, W. H.; PULLEYBLANK, W. R.; SCHRIJVER, A. *Combinatorial optimization*. New York: John Wiley, 1997.

COOPER, G. F.; HERSKOVITS, E. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, v. 9, p. 309–347, 1992.

COTTA, C. Protein structure prediction using evolutionary algorithms hybridized with backtracking. In: *International Work-Conference on Artificial and Natural Neural Networks: Part II: Artificial Neural Nets Problem Solving Methods*, IWANN '03, Berlin: Springer-Verlag, 2003, p. 321–328 (*IWANN '03*,).

CROCOMO, M. K.; DELBEM, A. C. B. *Otimização bayesiana com detecção de comunidades*. Relatório Técnico, Universidade de São Paulo, 2011.

DIESTEL, R. *Graph Theory*, v. 173 de *Graduate Texts in Mathematics*. 3 ed. Springer-Verlag, Heidelberg, 2005.
Disponível em: <http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/GraphTheoryIII.pdf> (Acessado em 08 fev. 2011)

DONETTI, L.; MUÑOZ, M. A. Detecting network communities: a new systematic and efficient algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, v. 2004, n. 10, p. P10012, 2004.

DUCH, J.; ARENAS, A. Community detection in complex networks using extremal optimization. *Physical Review E*, v. 72, n. 2, p. 027104+, 2005.

EFRON, B.; GONG, G. A leisurely look at the bootstrap, the jackknife, and cross-validation. *The Amer. Stat.*, v. 37, p. 36–48, 1983.

FOGEL, D. B. *Evolutionary computation : Toward a new philosophy of machine intelligence*. IEEE Press, 296 p., 2005.

GOLDBERG, D. E. *The design of innovation: Lessons from and for competent genetic algorithms*. Norwell: Kluwer Academic Publishers, 272 p., 2002.

HADI, A.; RASHIDI, F. Design of optimal power distribution networks using multiobjective genetic algorithm. In: *Advances in Artificial Intelligence*, New York: Springer, 2005, p. 203–215.

HARIK, G. R.; LOBO, F. G.; GOLDBERG, D. E. The compact genetic algorithm. *IEEE Trans. Evolutionary Computation*, v. 3, n. 4, p. 287–297, 1999.

HARIK, G. R.; LOBO, F. G.; SASTRY, K. Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga). In: *Scalable Optimization via Probabilistic Modeling*, p. 39–61, 2006.

HOOS, H. H.; STÜTZLE, T. *Stochastic local search: Foundations & applications*. Burlington: Elsevier: Morgan Kaufmann, 672 p., 2004.

JONG, K. A. D. *Evolutionary computation : A unified approach*. Cambridge: MIT Press, 256 p., 2006.

LIMA, T. W. D. *Algoritmos evolutivos para predição de estruturas de proteínas*. Dissertação de Mestrado, Universidade de São Paulo, São Carlos, 2006.

MANSOUR, M. R.; SANTOS, A. C.; J. B. A, L.; B., D. A. C.; BRETAS, N. G. Representação nó-profundidade e algoritmos evolutivos aplicados ao problema de restabelecimento de energia em sistemas de distribuição de energia elétrica. In: *Congresso Brasileiro de Automática*, Bonito, 2010, p. 1215 – 1221.

MARTINS, J. P.; SOARES, A. H. M.; VARGAS, D. V.; DELBEM, A. C. B. Multi-objective phylogenetic algorithm: Solving multi-objective decomposable deceptive problems. In: *6th international conference on Evolutionary Multi-criterion Optimization (EMO)*, Ouro Preto, MG - Brazil, 2011.

MELO, V. V.; DELBEM, A. C. Using smart sampling to discover promising regions and increase the efficiency of differential evolution. *International Conference in Intelligent Systems Design and Applications*, v. 0, p. 1394–1399, 2009.

MELO, V. V.; VARGAS, D. V.; CROCOMO, M. K.; DELBEM, A. C. B. Phylogenetic differential evolution. *IJNCR*, v. 2, n. 1, p. 21–38, 2011.

MOURA, A.; RIJO, R.; SILVA, P.; CRESPO, S. A multi-objective genetic algorithm applied to autonomous underwater vehicles for sewage outfall plume dispersion observations. *Applied Software Computing*, v. 10, n. 4, p. 1119–1126, 2010.

NEWMAN, M. E. J. Fast algorithm for detecting community structure in networks. 2003. Disponível em: <http://arxiv.org/abs/cond-mat/0309508> (Acessado em 09 fev. 2011)

NEWMAN, M. E. J.; GIRVAN, M. Finding and evaluating community structure in networks. *Physical Review E*, v. 69, n. 2, p. 026113+, 2004.
Disponível em: <http://dx.doi.org/10.1103/PhysRevE.69.026113> (Acessado em 09 fev. 2011)

NIEDERMAYER, D. An introduction to bayesian networks. 2009.
Disponível em: <http://www.niedermayer.ca/papers/bayesian/index.html> (Acessado em 09 fev. 2011)

OGATA, A. K. O. *Multialinhamento de seqüências biológicas utilizando algoritmos genéticos*. Dissertação de Mestrado, Universidade de São Paulo, São Carlos, 2006.

PEARL, J. *Probabilistic reasoning in intelligent systems:networks of plausible inference*. Burlington: Morgan Kaufmann, 552 p., 1988.

PELIKAN, M. *Hierarchical bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*. Studies in Fuzziness and Soft Computing, 1 ed. New York: Springer, 2005.

PELIKAN, M.; GOLDBERG, D. E.; CANTÚ-PAZ, E. Boa: The bayesian optimization algorithm. In: *Conference on Genetic and Evolutionary Computation*, Orlando: Morgan Kaufmann, 1999, p. 525–532.

PESSIN, G.; OSÓRIO, F. S.; WOLF, D. F.; BRASIL, C. R. S. Improving efficiency of a genetic algorithm applied to multi-robot tactic operation. In: MORALES, Á. F. K.; SIMARI, G. R., eds. *Advances in Artificial Intelligence, 12th Ibero-American Conference on AI, Bahía Blanca, Argentina, November 1-5, 2010. Proceedings*, Springer, 2010, p. 50–59 (*Lecture Notes in Computer Science*, v.6433).

RANA, S.; WHITLEY, D. Genetic algorithm behavior in the MAXSAT domain. Berlin: Springer, 1998, p. 785–794.

SANTOS, E. B. *A ordenação das variáveis no processo de otimização de classificadores bayesianos: Uma abordagem evolutiva*. Dissertação de Mestrado, Universidade Federal de São Carlos, São Carlos, 2007.

TALBI, E.-G. *Metaheuristics : from design to implementation*, v. 10 de *The Sciences Po series in international relations and political economy*. San Francisco: John Wiley & Sons, 1–6 p., 2009.

VARGAS, D. V.; DELBEM, A. C. B. *Algoritmo filogenético*. Relatório Técnico, Universidade de São Paulo, 2009.

YE, Z.; HU, S.; YU, J. Adaptive clustering algorithm for community detection in complex networks. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, v. 78, n. 4, p. 046115+, 2008.

Apêndice A

Exemplo de cálculo de métrica do K2

Para facilitar a compreensão do cálculo da métrica do algoritmo K2 apresentada no capítulo4, assim como as variáveis envolvidas, será demonstrado um exemplo utilizando a BN dada pela Figura A.1, e as soluções selecionadas dadas pela tabela A.1. Estes são os parâmetros de entrada B_s (a BN sendo avaliada) e D (o conjunto de dados que a BN deve representar) para o cálculo da métrica, repetida abaixo na equação A.1.

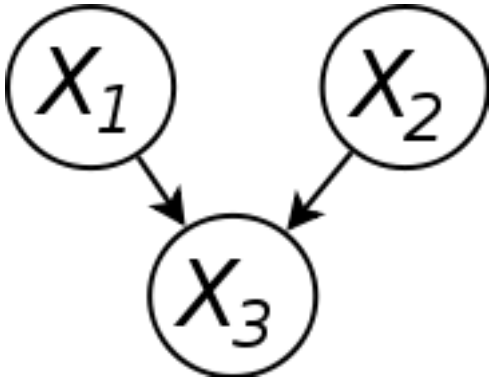

Figura A.1: Uma possível BN. Parâmetro de entrada B_s para cálculo da métrica do K2.

Tabela A.1: Soluções selecionadas. Parâmetro de entrada D para cálculo da métrica do K2.

X_1	X_2	X_3
1	0	1
1	0	0
1	0	1
1	1	1

$$P(B_s, D) = P(B_s) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!, \tag{A.1}$$

Encontrando os valores que serão utilizados na Equação A.1:

- n: Este parâmetro representa o número de variáveis do problema, neste exemplo, $n = 3$;
- r_i: Como neste problema as variáveis X_1, X_2 e X_3 são binárias, temos que $r_1 = r_2 = r_3 = 2$;
- q_i: Quantidade de instâncias para as variáveis pais de X_i. Como X_1 e X_2 não possuem variáveis pais, temos que $q_1 = q_2 = 0$. No entanto, como X_1 e X_2 são pais de X_3, ao observar a Tabela A.1 verifica-se que existem 2 possíveis instâncias para os pais de X_3: ($X_1 = 1$, $X_2 = 0$) e ($X_1 = 1$, $X_2 = 1$). Portanto temos $q_3 = 2$;
- w_{ij}: Instância j para as variáveis pais de X_i. Como visto no item acima, a única variável que possui variáveis pais é X_3, e as possíveis instâncias são: w_{31} = ($X_1 = 1$, $X_2 = 0$) e w_{32} = ($X_1 = 1$, $X_2 = 1$);
- N_{ij}: Número de ocorrências que possuem o conjunto de pais de X_i instanciados como w_{ij}. Ao observar a Tabela A.1, verifica-se que existem 3 ocorrências de w_{31} (referentes às 3 primeiras entradas, representadas pela Tabela A.2) e uma ocorrência para w_{32} (última entrada, representada na Tabela A.3). Desta forma, temos $N_{31} = 3$ e $N_{32} = 1$;
- N_{ijk}: Número de ocorrências em que os pais de X_i são instanciados como w_{ij}, e além disso, X_i possui a k-ésima possível instância. As possíveis instâncias para X_3 são $X_3 = 0$ (k=1) e $X_3 = 1$ (k=2). Considerando apenas as entradas da Tabela A.1 referentes à w_{31}, existe uma única entrada em que $X_3 = 0$ (representada na Tabela A.4), e duas entradas em que $X_3 = 1$ (representadas na Tabela A.5). Portanto, temos $N_{311} = 1$ e $N_{312} = 2$. Da mesma forma, considerando apenas a entrada referente à w_{32}, existem 0 entradas com $X_3 = 0$ e uma entrada com $X_3 = 1$ (única entrada representada na Tabela A.3). Portanto, temos $N_{321} = 0$ e $N_{322} = 1$.

Tabela A.2: Entradas referentes à w_{31} ($X_1 = 1$, $X_2 = 0$) representadas pela linha cinza.

X_1	X_2	X_3
1	0	1
1	0	0
1	0	1
1	1	1

Ao desenvolver o primeiro produtório da Equação(A.1), e admitindo $Y_{ij} = \frac{(r_i-1)!}{(N_{ij}+r_i-1)!} \prod_{k=1}^{r_i} N_{ijk}!$, temos:

$P(B_s, D) = (\prod_{j=1}^{q_1} Y_{1j}) * (\prod_{j=1}^{q_2} Y_{2j}) * (\prod_{j=1}^{q_3} Y_{3j})$,

Como $q_1 = q_2 = 0$:

$P(B_s, D) = \prod_{j=1}^{q_3} Y_{3j} = \prod_{j=1}^{q_3} \frac{(r_3-1)!}{(N_{3j}+r_3-1)!} \prod_{k=1}^{r_3} N_{3jk}!$,

Tabela A.3: Entradas referentes à w_{32} ($X_1 = 1$, $X_2 = 1$) representadas pela linha cinza.

X_1	X_2	X_3
1	0	1
1	0	0
1	0	1
1	1	1

Tabela A.4: Entrada referente à w_{31} ($X_1 = 1$, $X_2 = 0$) em que $X_3 = 0$ representada pela linha cinza.

X_1	X_2	X_3
1	0	1
1	0	0
1	0	1
1	1	1

Tabela A.5: Entradas referentes à w_{31} ($X_1 = 1$, $X_2 = 0$) em que $X_3 = 1$ representadas pela linha cinza.

X_1	X_2	X_3
1	0	1
1	0	0
1	0	1
1	1	1

Sendo $r_3 = 2$ e $q_3 = 2$, temos:

$P(B_s, D) = \frac{(2-1)!}{(N_{31}+2-1)!} * N_{311}! * N_{312}! * \frac{(2-1)!}{(N_{32}+2-1)!} * N_{321}! * N_{322}!$,

Substituindo pelos valores encontrados anteriormente:

$P(B_s, D) = \frac{(2-1)!}{(3+2-1)!} * 1! * 2! * \frac{(2-1)!}{(1+2-1)!} * 0! * 1!$,
$P(B_s, D) = \frac{1!}{4!} * 2 * \frac{1}{2} = \frac{1}{24}$.

Desta forma, temos que a pontuação da BN B_s ao representar os dados D é de 1/24, segundo a métrica utilizada no algoritmo K2.